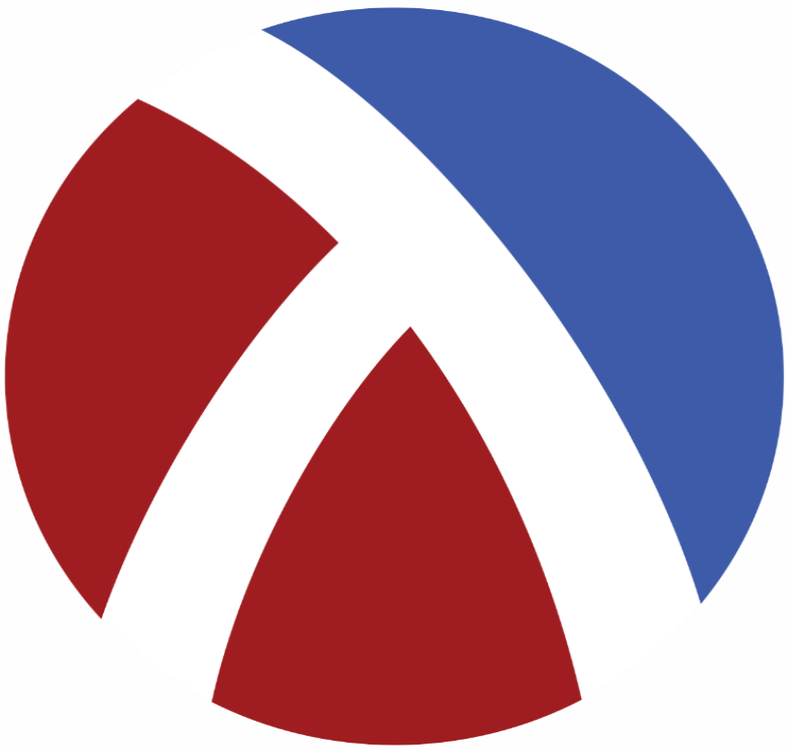
INFORME DE LABORATORIO 1: SIMULACION DE UN SISTEMA E INTERACCIÓN DE CHATBOTS EN DR. RACKET



Nombre: Reinaldo Pacheco Parra
Profesor: Gonzalo Martínez Ramírez
Asignatura: Paradigmas de programación (2/2023)

Índice

Introducción

En el siguiente informe, se presenta el desarrollo de un sistema de interacción de chatbots utilizando el Paradigma Funcional en el lenguaje Scheme a través del compilador Dr. Racket. Este informe se estructura en varias secciones clave, que abarcan la descripción del problema y los objetivos, la explicación del paradigma funcional, aspectos de análisis, diseño e implementación de la solución.
Se discute la razón por la cual se optó por una implementación en particular en lugar de otras alternativas y se destacan las ventajas de esta elección. A continuación, se exploran algunas de las funciones específicas que se han implementado y los enfoques que se tuvieron que considerar durante el proceso de desarrollo. Posteriormente, se proporcionan las instrucciones necesarias para verificar el correcto funcionamiento del programa.
Para concluir, se realiza un análisis de los resultados obtenidos a partir de la implementación y se lleva a cabo una autoevaluación detallada de cada una de las funciones implementadas en el sistema de interacción de chatbots.

Descripción breve del problema: Se busca realizar una simulación de un sistema de chatbots de ITR (interacción respuesta a texto) los cuales responden a preguntas en base a opciones especificas ya definidas por algún número o palabra clave que permite diferenciarlos entre sí. Este simulador de chatbots debe permitir realizar diferentes operaciones convencionales tales como: crear un flujo de opciones, crear un chatbot, agregar un flujo dentro de un chatbot, crear un sistema de chatbots, registrar usuarios para poder iniciar y cerrar sesión en el sistema, simular una conversación con un chatbot, entre otros.

Descripción breve del paradigma: La programación funcional es un paradigma de programación que trata la computación como la evaluación de funciones matemáticas, se basa en el cálculo lambda empleando funciones anónimas y evita que los estados o datos cambien, esto último se denomina inmutabilidad. En lugar de ejecutar secuencias de pasos como la programación imperativa, la programación funcional se enfoca en la evaluación de expresiones. La salida de una función está determinada únicamente por los parámetros de entrada, por ende, facilita el razonamiento sobre el código y su depuración. Además, este paradigma permite que las funciones sean utilizadas como argumentos o sean retornadas como valores en otras funciones, permitiendo el uso de la recursividad para construir operaciones con un mayor grado de complejidad.

Otros elementos importantes de este paradigma son los siguientes

Función Pura: Es una función, en donde su salida, depende únicamente de sus parámetros de entrada, esto quiere decir que, a mismos argumentos, se obtendrán los mismos resultados.

Función Anónima: Es una función que no está asociada a un nombre (símbolo). En Dr. Racket, se utiliza lambda para representar funciones anónimas. Estas funciones son útiles cuando se necesita pasar una función como argumento o retornarla sin necesidad de darle un nombre porque no se reutiliza en otro lugar.

Recursividad: Es una función la cual contiene una condición de termino, si esta condición no se cumple, la función se sigue ejecutando. Normalmente se utiliza un acumulador para la recursión de cola y un estado pendiente en el caso de la recursión natural. Cuando estos elementos

cumplen una condición determinada, la función termina y se entrega el resultado.

Currificación: La currificación se basa en convertir una función de n argumentos a una serie de funciones que contienen un solo argumento, luego de terminar una de las funciones, el resultado es ingresado a la siguiente y así sucesivamente. Esto facilita la composición de funciones, permitiendo trabajar con un elemento a la vez, esto permite dividir un problema extenso en subproblemas más simples.

Desarrollo

Análisis del problema

Se identifica que para la realización de un programa de simulación de un sistema de chatbots se necesitan los siguientes elementos:

- **System:** Un sistema tiene una representación en forma de lista que contiene un nombre de sistema, un id de chatbot inicial y un conjunto de 0 o más chatbots.
  (name (string) X InitialChatbotCodeLink (int) X chatbot)*
- **Chatbot:** Un chatbot tiene una representación en forma de lista que contiene un id de chatbot, un nombre de chatbot, un mensaje de bienvenida, un id de inicio de flujo y un conjunto de 0 o más flujos.
  (chatbotID (int) x name (string) x welcomeMessage (string) x startFlowId(int) x flows)*
- **Option:** Una opción tiene una representación en forma de lista que contiene un código de opción, un mensaje, un código de chatbot, un código inicial y un conjunto de 0 o más palabras claves.
  (code (int) x message (string) x ChatbotCodeLink (int) x InitialFlowCodeLink (int) x keyword)*
- **Flow:** Un flujo tiene una representación en forma de lista que contiene un código de flujo, un nombre y un conjunto de 0 o más opciones.
  (id (int) X name-msg (string) X option)*
- **User:** Un usuario tiene una representación en forma de lista que contiene el nombre del usuario.
  (username (string))
- **Chat-history:** Un historial de chat tiene una representación en forma de lista que almacena el historial de un usuario.
  (history (string))

Además, se deben realizar las siguientes operaciones entre los elementos:

option: crea una opción
flow: crea un flujo (conjunto de opciones)
flow-add-option: permite agregar una opción dentro de un flujo
chatbot: crea un chatbot.
chatbot-add-flow: permite agregar un flujo dentro de un chatbot
system: crear un sistema
system-add-chatbot: permite agregar un chatbot al sistema
system-add-user: permite agregar un usuario al sistema
system-login: inicia una sesión de un usuario en el sistema (el usuario debe estar registrado para poder iniciar sesión)
system-logout: cierra una sesión de un usuario en el sistema.
system-talk: permite interactuar con un chatbot.
system-synthesis: ofrece una síntesis del chatbot a partir del chat-history.
system-simulate: simula un diálogo entre dos chatbots en el sistema.

Diseño de la solución

El diseño de solución implementado, utilizando Programación Funcional consiste en representar los elementos del sistema de chatbots como listas. Esto se hace dado al buen manejo de Dr. Racket con las listas, además de que esta representación también permite agregar elementos dentro de otros. Por ejemplo, si queremos representar un flujo de opciones, debemos pensar cada opción como una lista, estos elementos estarán almacenados dentro de otra lista llamada flujo, y así sucesivamente con los demás elementos que conforman el sistema.

Uno de los enfoques principales de la problemática es que algunos de los elementos están representados por un id único que puede ser un entero o un string, lo cual evita que, al momento de agregar un elemento, no se tenga el mismo elemento dentro de la lista.

En el caso del chatbot, este posee un chatbotID (int), el flow posee un id (int) y el option posee un code (int) por lo que uno de los primeros desafíos corresponde a implementar un método para que los elementos fueran únicos y se diferenciaran de otros. Para esto, se aplicó una de las capas del TDA la cual es la capa de pertenencia para evitar duplicidad.

En una de las funciones **"flow-add-option"**, la cual permite agregar una opción a un flujo, se debe realizar esta verificación con el objetivo de no tener opciones duplicadas dentro del flujo. Para ello, se utiliza un condicional el cual pregunta si la opción se encuentra dentro del flujo, si no se cumple, se agrega dentro de la lista. *(ver figura 1 en Anexos)*

Otra de las funciones que utiliza este mismo método, pero utilizando recursión es la función **"chatbot-add-flow"** la cual actúa de la misma forma que la anterior, agrega flujos dentro de un chatbot y verifica que ese flujo agregado no esté dentro de la lista de chatbot, con el fin de evitar duplicidad. *(ver figura 2 en Anexos)*

La función **"system-add-user"** aplica la misma tendencia, evita que se agregue un usuario duplicado a un sistema, con la diferencia de que, en este caso, el usuario es representado por un string, por lo que se debe realizar el procedimiento de verificar que el string no se encuentre en la lista de usuarios antes de poder registrarlo en sistema. *(ver figura 3 en Anexos)*

Otra de las funciones fundamentales es la constructora **"system"** la cual construye un sistema, esto debido a que la gran parte de los elementos del proyecto se deben almacenar en esta lista. Para la implementación se agregaron elementos adicionales en el sistema siendo estos: la hora de creación del sistema, una lista de usuarios registrados, una lista de usuario que tiene una sesión activa y una lista que almacena el historial del chat del sistema. *(ver figura 4 en Anexos)*

Aspectos de implementación

Para la resolución del problema, se utiliza el Paradigma Funcional mediante el lenguaje de programación Scheme en el compilador Dr. Racket de versión 8.9 sin el uso de bibliotecas externas, además se prohíbe el uso del método "set!" el cual permite modificar elementos ya que este rompe con la definición de Paradigma Funcional. Para la implementación se tiene una representación con TDA para cada elemento ya que permite organizar y ordenar de mejor forma los elementos. Por último, la construcción de la mayoría de las funciones se distribuyeron en capas las cuales son:

- **Constructor:** Crea la estructura o representación. Se representa con la palabra “make”.
- **Pertenencia:** Valida operaciones o reglas, entrega un booleano. Se representa con “is?” o “exists?”
- **Selector:** Obtiene datos específicos de una representación. Se representa con la palabra “get”.
- **Modificador:** Permite modificar datos de la representación. Se representa con “set”.
- **Otras funciones**

Los TDA utilizados en la implementación fueron:

- **TDA System**
- **TDA Chabot**
- **TDA Flow**
- **TDA Option**
- **TDA User**
- **TDA ChatHistory**

(ver Tablas en Anexo donde se adjuntan todas las funciones implementadas por nombre, capa, tipo de recursión y descripción)

Instrucciones de uso

Ejemplos de uso

Para el correcto uso de este programa, se deben tener todos los archivos llamados “TDA” y los archivos de prueba dentro de la misma carpeta. Esto con el fin de que compartan la misma ubicación para evitar errores al momento de hacer llamadas a funciones entre los TDA.

El primer archivo llamado pruebas1 contiene el primer script de pruebas el cual nos permite crear opciones, flujos, chatbots y un sistema, además de insertar opciones dentro de un flujo, flujos dentro de un chatbot e insertar chatbots dentro de un sistema. También permite registrar, iniciar sesión y cerrar sesión de un usuario. En la consola se puede hacer un llamado a cualquiera de estas funciones para verificar el correcto funcionamiento de estas. En el archivo de pruebas2 se encuentra un script más detallado el cual permite verificar las mismas operaciones, además de construir el sistema que será utilizado para realizar otros procedimientos más complejos como interactuar o hablar con un chatbot.

Resultados Esperados

Se espera que el programa logre crear correctamente un sistema completo el cual contiene una lista de chatbots, a su vez, cada chatbot contiene un conjunto de flujos y cada flujo tiene las opciones correspondientes. Permitir registrar e iniciar sesión a usuarios y guardar el historial de chat de la conversación, además de realizar interacciones permitiendo al chatbot entregar respuesta dependiendo de las opciones solicitadas por el usuario.

Posibles errores

Para la función system-talk-rec se tomaron algunos supuestos, por ejemplo, que el primer chatbot que inicia dentro del sistema ante la primera interacción que realiza el usuario es el chatbot inicial, el cual está detallado según el script de prueba, si no se toma este supuesto, es probable que la conversación pierda sentido ya que el chatbot podría entregar un flujo de opciones directamente antes de realizar un mensaje de bienvenida hacia el usuario.

Resultados y autoevaluación

Resultados

Las funciones implementadas se probaron de múltiples formas para verificar su correcto funcionamiento. Se verificaron mediante los 2 script de prueba disponibles, además de crear otro script con la misma estructura, pero con

algunos elementos modificados, funcionando correctamente para todos obteniendo las respuestas y resultados esperados.
Las funciones system-simuate y system-sinthesis no se lograron implementar correctamente debido a la complejidad de estas.

Autoevaluación

Las funciones flow, flow-add-option, chatbot, chatbot-add-flow, system, system-add-chatbot, system-add-user, system-login, system-logout fueron implementadas en su totalidad funcionando correctamente para ambos script y pruebas realizadas. Las funciones system-talk-rec y system-talk-norec fueron implementadas parcialmente.
Por último, las funciones system-simuate y system-sinthesis no fueron implementadas.

Conclusión

Después de llevar a cabo el programa de simulación de un sistema de chatbots utilizando el Paradigma Funcional en Dr. Racket, es evidente que se lograron satisfacer la mayoría de los requisitos planteados en la problemática. Inicialmente, se enfrentó el desafío de comprender y adaptarse al paradigma funcional, dado que previamente se había trabajado principalmente con el paradigma imperativo en otras asignaturas. Sin embargo, a medida que se avanzó en la implementación de los requisitos, se observó que algunas funciones podían ser implementadas de manera bastante sencilla en tan solo 2 o 3 líneas, mientras que otras requerían ser desglosadas en subproblemas para llevar a cabo operaciones específicas.

Fue crucial abordar el problema de manera adecuada y trabajar con listas dentro listas, ya que elegir otra representación, como la representación en forma de árbol, habría complicado significativamente la construcción de la solución. Además, el uso de la capa de Tipos de Datos Abstractos (TDA) permitió una mejor organización de la problemática al posibilitar la creación de funciones selectoras, de pertenencia y constructoras que podían ser llamadas en múltiples funciones, reduciendo considerablemente la cantidad de código en algunas de ellas y permitiendo una mejor organización e implementación del proyecto.

Bibliografía y referencias

Racket Documentation (n.d) extraído de: https://docs.racket-lang.org

González Ibáñez, R. (2023). Proyecto Semestral de Laboratorio Paradigmas de Programación 2023/2
https://docs.google.com/document/d/1LB2b3J71Baqa_IuZt6EmRwDlxC0qzWn9YJAm6FIiJk/edit

González Ibáñez, R. (2023). Proyecto Semestral de Laboratorio General 2023/2
https://docs.google.com/document/d/1L-B2b3J71Baqa_IuZt6EmRwDlxC0qzWn9YJAm6FIiJk/edit

Anexos

```
; 4. TDA Flow - modificador
; Nombre funcion: flow-add-option
; Dominio: flow x option
; Recorrido: flow
; Recursión: no aplica
; Descripción: Funcion modificadora para añadir opciones a un flujo

(define (flow-add-option new-flow option)
  (if (option-exists? option (get-options new-flow))
      new-flow
      (new-flow (get-id new-flow) (get-name-msg new-flow) (append (get-options new-flow) (list option)))))
```

Figura 1: Función flow-add-option

```
; 6. TDA chatbot - modificador
; Nombre función: chatbot-add-flow
; Dominio: list X list
; Recorrido: list
; Recursión: cola
; Descripción: Función modificadora para añadir flujos a un chatbot.
(define (chatbot-add-flow chatbot flow)
  (let ([flows (get-flows chatbot)])
    (define (aux-flow-adder flows flow)
      (cond
        [(null? flows) (list flow)]
        [(flow-exists? flow flows) flows]
        [else (append flows (list flow))]))
    (list (get-chatbotID chatbot)(get-name-chatbot chatbot) (get-welcomeMessage chatbot) (get-startFlowId chatbot)(aux-flow-adder flows flow))))
```

Figura 2: Función chatbot-add-flow

```
; 9. TDA system - modificador
; Nombre de la función: system-add-user
; Dominio: system x user (string)
; Recorrido: system
; Recursión: no aplica
; Descripción: Función modificadora para añadir usuarios a un sistema.
(define (system-add-user system user)
    (if (not (member user (get-existing-users system)))
        (list (get-system-name system)
              (get-system-initialChatbotCodeLink system)
              (get-system-creation-time system)
              (get-system-chatbots system)
              (cons user (get-existing-users system))
              (get-system-logged-in-user system)
              (get-system-chat-history system))
        system))
```

Figura 3: Función system-add-user

```
; 7. TDA system - constructor
; Dom: name (string) X initialChatbotCodeLink (int) X chatbot* (puede recibir 0 o más chatbots)
; Recorrido: list
; Recursión: no aplica
; Descripción: Función constructora de un sistema de chatbots. Deja registro de la fecha de creación.

(define (system name InitialChatbotCodeLink . chatbots)
  (let ([unique-chatbots (remove-duplicates chatbots)])
    (list name InitialChatbotCodeLink (current-seconds) unique-chatbots '() '() '())))
```

Figura 4: Función constructora de un system

Tabla 1: TDA Option

Nombre	Capa	Recursión	Descripción
option	Constructor	No aplica	Crea una opción
get-code	Selector	No aplica	Obtiene el código de la opción
get-message	Selector	No aplica	Obtiene el mensaje de la opción
get-chatbotcodelink	Selector	No aplica	Obtiene el code de unión con el chatbot
get-initialflowcodelink	Selector	No aplica	Obtiene el código de flujo
get-keywords	Selector	No aplica	Obtiene las palabras claves de una opción

Tabla 2: TDA Flow

Nombre	Capa	Recursión	Descripción
flow	Constructor	No aplica	Construye un flujo
remove-duplicates-by-id	Otras funciones	Cola	Elimina los duplicados por id
get-id	Selector	No aplica	Obtiene el id de un flujo
get-name-msg	Selector	No aplica	Obtiene el mensaje de un flujo
get-options	Selector	No aplica	Obtiene la lista de opciones de un flujo
option-exists?	Pertenencia	No aplica	Verifica si una opción existe dentro de un flujo
flow-add-option	modificador	No aplica	Añade opciones a un flujo

Tabla 3: TDA Chatbot

Nombre	Capa	Recursión	Descripción
chatbot	Constructor	No aplica	Crea un chatbot
remove-duplicates	Otras funciones	Cola	Elimina los flujos duplicados en base a su id
get-chatbotID	Selector	No aplica	Obtiene el id del chatbot
get-name-chatbot	Selector	No aplica	Obtiene el nombre del chatbot
get-welcomeMessage	Selector	No aplica	Obtiene el mensaje de bienvenida del chatbot
get-startFlowId	Selector	No aplica	Obtiene el id de flujo inicial del chatbot
get-flows	Selector	No aplica	Obtiene los flujos del chatbot
flow-exists?	Pertenencia	No aplica	Verifica si ya existe el flujo en el chatbot
chatbot-add-flow	Modificador	Cola	Añade flujos a un chatbot

Tabla 4: TDA System

Nombre	Capa	Recursión	Descripción
system	Constructor	No aplica	Crea un sistema
get-system-name	Selector	No aplica	Obtiene el nombre del sistema
get-system-initialchatbotcodelink	Selector	No aplica	Obtiene el código inicia de chatbot del sistema
get-system-creation-time	Selector	No aplica	Obtiene la hora de creación del sistema
get-system-chatbots	Selector	No aplica	Obtiene la lista de chatbots del sistema
get-existing-users	Selector	No aplica	Obtiene la lista de usuarios registrados
get-logged-in-user	Selector	No aplica	Obtiene una lista con el usuario logeado
get-system-chat-history	Selector	No aplica	Obtiene el historial de chat
get-current-id-chatbot	Selector	No aplica	Obtiene el id del chat actual en el sistema
chatbot-exists?	Pertenencia	No aplica	Verifica si un chatbot existe en el sistema
user-exists?	Pertenencia	No aplica	Verifica si un usuario existe en el sistema
system-add-chatbot	Modificador	No aplica	Añade un chatbot al sistema
system-add-user	Modificador	No aplica	Registra un usuario en el sistema
system-login	Modificador	No aplica	Logea un usuario en el sistema
system-logout	Modificador	No aplica	Cierra la sesión de un usuario en el sistema

search-chatbot	Otras funciones	No aplica	Permite obtener un chatbot según el mensaje
search-chatbot-flow	Otras funciones	No aplica	Permite obtener un flujo según el mensaje
search-chatbot-chat	Otras funciones	Natural	Busca el chat de un chatbot
system-talk-rec	Otras funciones	Natural	Simula una interacción con un chatbot
search-chatbot-id	Selector	No aplica	Obtiene el id del chatbot actual
search-flow-id	Selector	No aplica	Obtiene el id del flujo actual
list-id	Otras funciones	No aplica	Crea una lista con el id del chatbot e id de flujo actuales
System-talk-norec	Otras funciones	No aplica	Simula una interacción con un chatbot

Tabla 5: TDA User

Nombre	Capa	Recursión	Descripción
make-user	Constructor	No aplica	Crea un usuario
get-username	Selector	No aplica	Obtiene el nombre de un usuario

Tabla 6: TDA ChatHistory

Nombre	Capa	Recursión	Descripción
Chat-history	Constructor	No aplica	Crea el historial de chat